Problemas del Tema 1

1.1. Convertir a base decimal los siguientes números:

- $(201.2)_3$

Para pasar a base decimal tenemos que utilizar el método polinómico, por lo que:
$N = 2*3^2+0*3^1+1*3^0+2*3^{-1}=\mathbf{21.667}$

- $(FFA.7)_{16}$

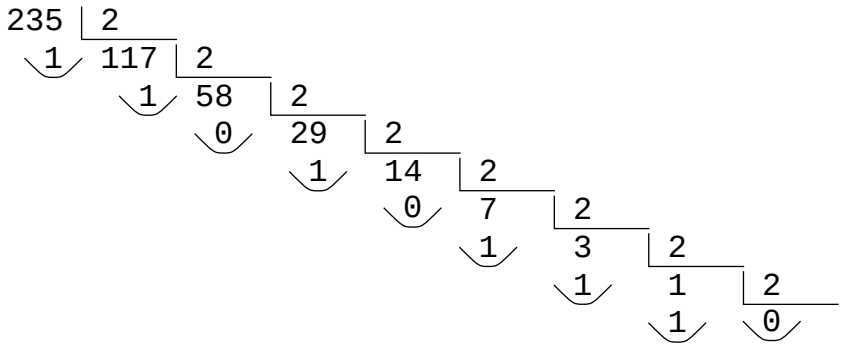
Como estamos en la base hexadecimal, las letras tiene un equivalente numérico (el que se utilizará en el polinomio): $N = 15*16^2+15*16^1+10*16^0+7*16^{-1}=\mathbf{4090.4375}$

- $(100)_5$
- $(26.5)_7$
- $(326.5)_9$

1.2. Convertir a base binaria los siguientes números:

- $(235.3)_{10}$

Para pasar a binario, debemos utilizar el método iterativo, para lo cual hay que separar la parte entera de la parte decimal. La conversión de la parte entera es:

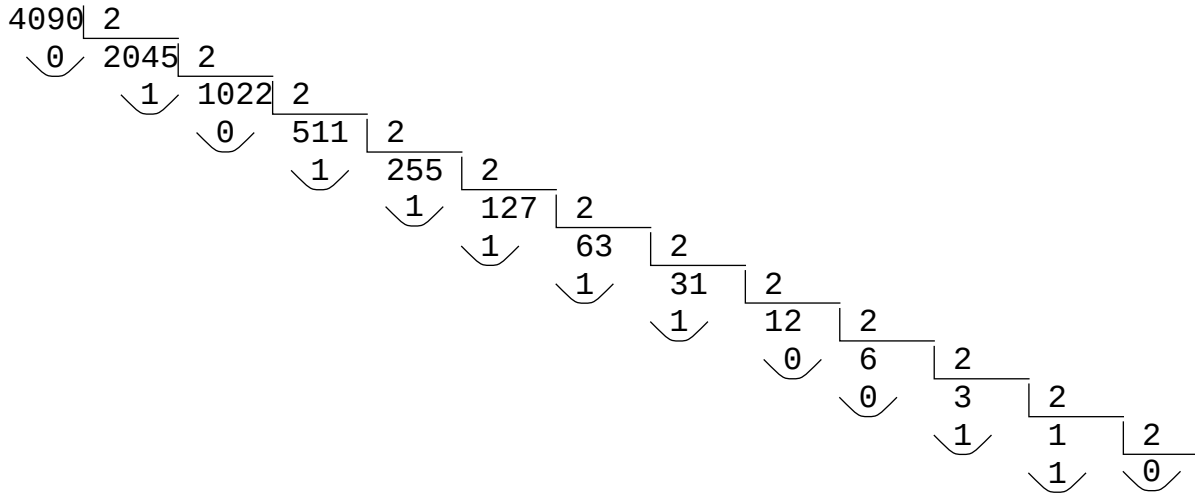


Mientras que la de la parte decimal es:
$0.3*2=\mathbf{0}.6$
$0.6*2=\mathbf{1}.2$
$0.2*2=\mathbf{0}.4$
$0.4*2=\mathbf{0}.8$
$0.8*2=\mathbf{1}.6$
Por lo tanto el número binario será: **011101011.01001**

- $(FFA.7)_{16}$

Como no sabemos realizar las operaciones aritméticas en base hexadecimal, primero debemos pasar dicho número a decimal y después a binario. El número deci-

mal será: N=4090.4375. Ahora utilizamos el método iterativo, esparando la parte entera de la parte decimal.



0.4375*2=**0**.8750
0.8750*2=**1**.7500
0.7500*2=**1**.5000
0.5000*2=**1**.0000
Por lo tanto, el número binario será: **0110011111010.0111**

- $(100)_8$
- $(26.5)_7$
- $(210.1)_3$

1.3. Convertir directamente:

- $(340)_8$ al sistema binario

Para pasar directamente a binario, la base fuente debe ser un potencia de dos. Entonces, se convierte dígito a dígito con tantos bits como dicha potencia. En este caso la potencia es 3; así '3'='011', '4'='100' y '0'='000'. De ahí el número binario será: **011100000**

- $(100)_8$ al sistema binario
- $(1B4)_{16}$ al sistema binario

En este caso la potencia es 4. Por lo tanto: '1'='0001', 'B'='1011', '4'='0100'. Luego, el número binario será: **000110110100**

- $(1000110)_2$ al sistema octal

Para pasar desde binario a una base que es potencia de dos, se agrupan los bits en dicha potencia (empezando

por el punto decimal). Como en este caso es la base octal, los grupos serán de tres bits, es decir, '1''000''110'. Ahora convertimos cada grupo por separado, por lo que número octal será: **106_8**

- $(1000110)_2$ al sistema hexadecimal

- $(1110110)_2$ al sistema hexadecimal

En este caso, los grupos serán de cuatro bits por estar en la base hexadecimal. De ahí, los grupos serán: '111''0110'. Entonces el número hexadecimal será: **36_{16}**

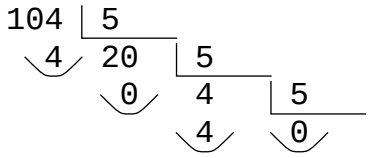
1.4. Dado la siguiente igualdad: $(100)_{10} = (400)_b$, determinar el valor de la base b. ¿Cuál es el valor de $(104)_{10}$ en la base b?

Como conocemos los valores de un número en decimal y en la base desconocida, podemos plantear la onversión polinómica:

$100 = 4*b^2+0*b^1+0*b^0$

De dicha ecuación podemos obtener el valor de la base, b = 5.

Para pasar 104 decimal a base 5, debemos utilizar el método iterativo



Por lo tanto, el número será: **0404_5**

1.5. Dado un código pesado con los siguientes pesos: (120,60,20,5,1), con la siguiente expresión polinómica:

$N^* = (n_4\ n_3\ n_2\ n_1\ n_0)^* = 120*n_4 + 60*n_3 + 20*n_2 + 5*n_1 + n_0$

Donde: $0 < n_0 < 4 \quad 0 < n_1 < 3 \quad 0 < n_2 < 2 \quad 0 < n_3 < 1 \quad 0 < n_4 < 1$

Se desea averiguar:

- El número decimal correspondiente a la palabra de código (01212)*

Como tenemos el polinomio de dicha base, para pasar a decimal, sólo tenemos que incluir los dígitos y realizar las operaciones: **N=120*0+60*1+20*2+5*1+2=107**

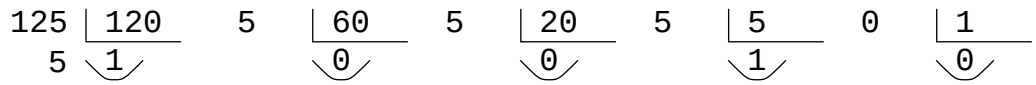
- El número decimal correspondiente a la palabra de código (10010)*

- La palabra de código correspondiente al número decimal 125

Para pasar de decimal a la nueva base debemos utilizar el método iterativo. Ahora podemos actuar de dos formas diferentes.

En primer lugar podemos dividir entre el mayor peso. De esta forma sabemos que el cociente es el dígito

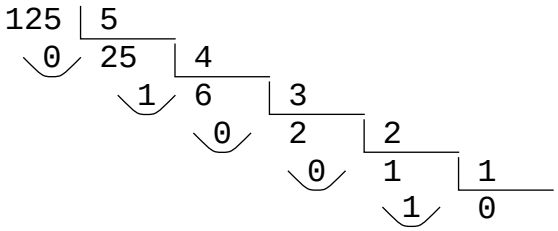
correspondiente, y el resto será la combinación del resto de dígitos. Se repite este método con todos los pesos de mayor a menor.



Por lo que el número será: **(10010)*.** Este método no es muy utilizado ya que, por lo general, no conocemos el mayor peso del numero.
En segundo lugar, debemos encontrar una relación entre los diferentes pesos de las bases:

- 5/1=5
- 20/5=4
- 60/20=3
- 120/60=2

Es decir, el polinomio se puede poner de la siguiente forma: $N^* = (n_4\ n_3\ n_2\ n_1\ n_0)^* = (((2*n_4 + n_3)*3 + n_2)*4 + n_1)*5 + n_0$. Por lo tanto, si dividimos entre 5, 4, 3, 2 y 1, los restos serán los dígitos menos significativos:

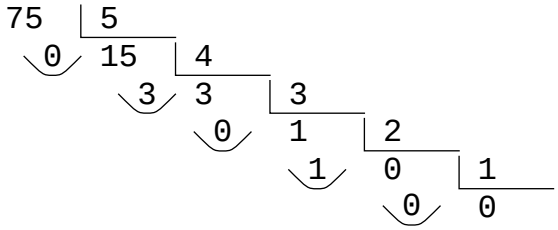


Por lo tanto, el número será: **(10010)***
Ambos métodos son igualmente válidos

- La palabra de código correspondiente al número decimal 230

- La palabra de código correspondiente al número binario (1001011)

Como la base no es una potencia de dos, no se puede pasar directamente, así que primero debemos pasr el número binario a decimal, y después a la nueva base. Para pasarlo a decimal utilizamos el método polinómico: $N=1*2^6+1*2^3+1*2^1+1*2^0=75$
Ahora utilizamos el método iterativo



Por lo tanto, el número representado en la nueva base será: **(01030)***

1.6. Dados los siguientes códigos BCD:

Dígito decimal	Código nº 1
0	0 0 0 0 0 0 0
1	0 0 0 1 1 0 1
2	0 0 1 1 0 1 1
3	0 0 1 0 1 1 0
4	0 1 1 0 0 0 1
5	0 1 1 1 1 0 0
6	0 1 0 1 0 1 0
7	0 1 0 0 1 1 1
8	1 1 0 0 1 0 0
9	1 1 0 1 0 0 1

Se pide:

- Razonar si es un código detector y/o corrector de errores (1 solo bit erróneo)

Para que el código sea detector (corrector) de errores en un solo bit, la distancia entre todas sus palabras debe ser de al menos de 2 (3). Por lo tanto, tenemos que hallar la distancia entre todas las palabras:

- d(0,1)=3, d(0,2)=4, d(0,3)=3, d(0,4)=3, d(0,5)=4, d(0,6)=3, d(0,7)=4, d(0,8)=3, d(0,9)=4, d(1,2)=3, d(1,3)=4, d(1,4)=4, d(1,5)=3, d(1,6)=4, d(1,7)=3, d(1,8)=4, d(1,9)=3, d(2,3)=3, d(2,4)=3, d(2,5)=4, d(2,6)=3, d(2,7)=4, d(2,8)=7, d(2,9)=4, d(3,4)=4, d(3,5)=3, d(3,6)=4, d(3,7)=3, d(3,8)=4, d(3,9)=7, d(4,5)=3, d(4,6)=4, d(4,7)=3, d(4,8)=4, d(4,9)=3, d(5,6)=3, d(5,7)=4, d(5,8)=3, d(5,9)=4, d(6,7)=3, d(6,8)=4, d(6,9)=3, d(7,8)=3, d(7,9)=4, d(8,9)=3.

Como la distancia mínima entre todas sus palabras es igual a 3, este código es un código corrector de errores en un solo bit (y por tanto también es detector)

- Detectar y/o corregir (en el caso que sea posible) las siguientes palabras de código. Para ello se debe indicar la particularidad del código detector utilizada para la detección, o los bits de mensaje que son chequeados por cada bit de cheueo.
  - 0000001
  - 1100100
  - 0000000
  - 1101011
  - 0000111
  - 0000110

(En el caso de que el código sea corrector, se sabe que los cuatro primeros bits son bits de mensaje y el resto bit de chequeo con paridad de tres bits)

Como el código es un código corrector de errores en un solo bit, vamos a indicar los bits de mensaje (los tres primeros) que son chequeados por cada bit de chequeo (los tres últimos). Nombrémos los bits de la siguiente forma: m1m2m3m4c1c2c3

Para ello, vamos a fijarnos en las palabras que sólo tienen un ‘1’ en los bits de mensaje. De esta forma los bits de chequeo que valgan ‘1’, chequearán a dicho bit. Utilizando este criterio, observamos que:

- c1 chequea a m4, m3, m2
- c2 chequea a m3, m2
- c3 chequea a m4, m2

Como no hay ninguna palabra que tenga un ‘1’ en m1 y ‘0’ en los demás bits de mensaje, pasamos a comprobar las palabras con dos 1’s. Los bits de chequeo que valgan ‘0’, chequearán a ambos bits; obteniendo:

- c2 chequea a m1
- c3 chequea a m1

Unificando ambas tablas, obtenemos la tabla de chequeo completa:

- c1 chequea a m4, m3, m2
- c2 chequea a m3, m2, m1
- c3 chequea a m4, m2, m1

Para el caso de la corrección, debemos obtener el valor de los bits de chequeo correspondientes a los bits de mensajes que nos han llegado, y compararlos con los bits de chequeo que nos han llegado. En el caso de que sean los mismos, la palabra obtenida es la correcta.

En el caso de la palabra 0000001, los bits de chequeo c1, c2 y c3 deberían valer ‘0’, pero c3 vale ‘1’. Esta situación nos indica que ha existido un error. Ahora debemos decidir en qué bit se ha producido el error. Como el error se ha detectado en c3, los bits que pueden ser erróneos son: c3, m4, m2 y m1. Además sabemos que el error se ha producido en un solo bit. Luego,

- si se hubiese producido en m4, el error también se hubiera detectado en c1;
- si se hubiese producido en m2, también se hubiese detectado en c1 y c2;
- y si se hubiese producido en m1, también se hubiese detectado en c2.

Por lo tanto, el error debe estar en el propio bit de chequeo c3. Así la palabra correcta sería: **0000000**

En el caso de la palabra 1100100, los bits de chequeo c1, c2 y c3 deberían valer ‘100’, como los bits de chequeo que han llegado. Por lo tanto, la palabra recibida es **correcta**.

En el caso de la palabra 0000111, los bits de chequeo c1, c2 y c3 deberían valer ‘0’, pero c2 y c3 vale ‘1’.

Esta situación nos indica que ha existido un error. Ahora debemos decidir en qué bit se ha producido el error. Como el error se ha detectado en c1, c2 y c3, los bits que pueden ser erróneos son: m2 (puesto que un error en algún otro implicaría un error en más de un bit). Así la palabra correcta sería: **0100111**

- Razonar una posible variante para obtener un código para detectar fallos de dos bits a partir de un código corrector de error de un bit.

Un método para obtener un código detector consiste en añadir un bit de paridad total. Pero para que este método surta efecto, debemos asegurarnos que la paridad de las palabras del código original no es la misma en todas las palabras. Esta situación se cumple en nuestro código ya que hay palabras con 0, 3 y 4 1's. Por lo tanto, añadimos un bit de paridad total. Así si se detecta un fallo, en los bits de chequeo y de paridad, el error está en un solo bit; mientras que si el fallo sólo se detecta en los bits de chequeo, se ha producido un fallo en dos bits.

1.7. Repetir el ejercicio anterior con los siguientes códigos.:

Dígito decimal	Código nº 1	Código nº 2	Código nº 3	Código nº 4
0	0 0 0 0 0 0 1	0 0 1 1 1 1 0	0 1 1 1 1 0 1	0 0 0 0 0 0 0
1	0 0 0 0 1 1 1	0 1 0 0 1 1 0	1 0 1 1 1 0 1	0 1 1 1 1 0 0
2	0 0 1 1 1 1 1	0 1 0 1 1 0 1	1 1 0 1 1 0 1	0 1 1 0 0 1 1
3	1 1 1 1 1 1 1	0 1 1 0 0 1 1	1 1 1 0 1 0 1	0 1 0 1 0 1 0
4	0 0 0 0 0 1 0	0 1 1 1 0 0 0	1 1 1 1 0 0 1	0 1 0 0 1 0 1
5	0 0 0 1 1 1 0	1 0 0 0 1 1 1	0 1 1 1 1 1 0	1 0 1 1 0 1 0
6	0 1 1 1 1 1 0	1 0 0 1 1 0 0	1 0 1 1 1 1 0	1 0 1 0 1 0 1
7	0 0 0 0 1 0 0	1 0 1 0 0 1 0	1 1 0 1 1 1 0	1 0 0 1 1 0 0
8	0 0 1 1 1 0 0	1 0 1 1 0 0 1	1 1 1 0 1 1 0	1 0 0 0 0 1 1
9	1 1 1 1 1 0 0	1 1 0 0 0 0 1	1 1 1 1 0 1 0	1 1 1 1 1 1 1

Problemas del Tema 2

2.1. Usando los postulados del Álgebra de Boole y los teoremas asociados, demuestre la veracidad de las siguientes igualdades:

- $x \cdot y + x' \cdot y' + x \cdot y' = x + y'$

Utilizando el teorema de idempotencia, y reordenando la expresión podemos poner:
xy+xy'+xy'+x'y'
A continuación sacamos factor común:
x(y+y')+(x+x')y'
Aplicando el cuarto postulado del Álgebra de Boole:
x+y'
Por lo tanto, **la igualdad es correcta.**

- $(x' \cdot z' + x' \cdot y + x' \cdot z + x \cdot y)' = x \cdot y'$

Si reordenamos la expresión, obtenemos:
(x'z'+x'z+x'y+xy)'
Sacando factor común:
(x'(z+z')+(x'+x)y)'
Aplicando el cuarto postulado:
(x'+y)'
Aplicando las leyes de DeMorgan:
xy'. Por lo tanto, **la igualdad es correcta**.

- $(x + y) \cdot (x' \cdot z' + z) \cdot (y' + x \cdot z') = x' \cdot y$

Si aplicamos el teorema 7 en el paréntesis central, obtenemos:
(x+y)(x'+z)(y'+xz')
Multiplicando los dos primeros paréntesis:
(xz+x'y+yz)(y'+xz')
Multiplicando los paréntesis que nos quedan:
xzy'. Por lo tanto, **la igualdad es falsa.**

- $x \cdot y + y \cdot z + x' \cdot z = x \cdot y + x' \cdot z$

- $(x + y) \cdot (x' + z) = x \cdot z + x' \cdot y$

Multiplicando los paréntesis, obtenemos:
xz+x'y+yz
Si multiplicamos por 1=x+x', el término que queremos eliminar, yz
xz+x'y+yz(x+x') Multiplicando el paréntesis:
xz+x'y+xyz+x'yz
Reordenando los términos y sacando factor común:
xz(1+y)+x'y(1+z)
Utilizando el teorema de los elementos nulos:
xz+x'y. Por lo que **la igualdad es correcta.**

- $x \cdot y + y \cdot z + x \cdot z = (x + y) \cdot (y + z) \cdot (x + z)$

- $x \cdot y' + y \cdot z' + x' \cdot z = x' \cdot y + y' \cdot z + x \cdot z'$

2.2. Pruebe que en un Álgebra de Boole, se verifican las siguientes leyes de cancelación:

- Si a+b=a+c y a'+b=a'+c, entonces b = c

Si multiplicamos los términos de las dos expresiones de partida, obtenemos:
(a+b)(a'+b)=(a+c)(a'+c)
ab+a'b+b=ac+a'c+c
Utilizando el teorema 7, obtenemos:
ab+b=ac+c
Si, de nuevo, utilizamos el mismo teorema, obtenemos:
b=c
Por lo tanto, **se verifica la ley de cancelación.**

- Si a+b=a+c y a·b=a·c, entonces b = c

2.3. Determinar si el conjunto B={0,a,b,1} y las operaciones (+) y (·) definidas como:

+	0	A	B	1	·	0	A	B	1
0	0	A	B	1	0	0	0	0	0
A	A	A	1	1	A	0	A	0	A
B	B	1	B	1	B	0	0	B	B
1	1	1	1	1	1	0	A	B	1

es un álgebra de Boole.

Para demostrar que se trata de un álgebra de Boole, se deben cumplir los cuatro postulados de dicho algebra.

- Operaciones conmutativas. La comprobación de este postulado es visual, ya que las tablas son simétricas respecto de su diagonal.
- Operaciones distributivas. Para ello, construimos las tablas de verdad de que parte de la igualdad de dichas propiedades. Estas tablas se muestran en la siguiente página.
- Elementos neutros diferentes. Podemos observar de las tablas que el elemento neutro de la suma es el '0', y el del producto es el '1'.
- Elementos complementos. La forma más fácil de demostrar este postulado es encontrar los complementos de todos los elementos de B, es decir, parejas de elementos que cumplan XY=0 y X+Y=1. Estas parejas son:
  - A'=B
  - 0'=1
  - B'=A
  - 1'=0

2.4. Obtenga los complementos de las siguientes funciones, así como las tablas de combinaciones y sus fórmulas canónicas disyuntivas y conjuntivas (tanto del complemento obtenido como de la función original):

- F = a + b·c

X	Y	Z	X(Y+Z)	XY+XZ	X+YZ	(X+Y)(X+Z)
A	A	A	A	A	A	A
		B	A	A	A	A
		0	A	A	A	A
		1	A	A	A	A
	B	A	A	A	A	A
		B	0	0	1	1
		0	0	0	A	A
		1	A	A	1	1
	0	A	A	A	A	A
		B	0	0	A	A
		0	0	0	A	A
		1	A	A	A	A
	1	A	A	A	A	A
		B	A	A	1	1
		0	A	A	A	A
		1	A	A	1	1
B	A	A	0	0	1	1
		B	B	B	B	B
		0	1	1	B	B
		1	B	B	1	1
	B	A	B	B	B	B
		B	B	B	B	B
		0	B	B	B	B
		1	B	B	B	B
	0	A	0	0	B	B
		B	B	B	B	B
		0	0	0	B	B
		1	B	B	B	B
	1	A	B	B	1	1
		B	B	B	B	B
		0	B	B	B	B
		1	B	B	1	1

X	Y	Z	X(Y+Z)	XY+XZ	X+YZ	(X+Y)(X+Z)
0	A	A	0	0	A	A
		B	0	0	0	0
		0	0	0	0	0
		1	0	0	A	A
	B	A	0	0	0	0
		B	0	0	B	B
		0	0	0	0	0
		1	0	0	B	B
	0	A	0	0	0	0
		B	0	0	0	0
		0	0	0	0	0
		1	0	0	0	0
	1	A	0	0	A	A
		B	0	0	B	B
		0	0	0	0	0
		1	0	0	1	1
1	A	A	A	A	1	1
		B	1	1	1	1
		0	A	A	1	1
		1	1	1	1	1
	B	A	1	1	1	1
		B	B	B	1	1
		0	B	B	1	1
		1	1	1	1	1
	0	A	A	A	1	1
		B	B	B	1	1
		0	0	0	1	1
		1	1	1	1	1
	1	A	1	1	1	1
		B	1	1	1	1
		0	1	1	1	1
		1	1	1	1	1

Para obtener el complemento de la función, aplicamos las leyes de DeMorgan:
F'= a'(b'+c')
A continuación, vamos a obtener la tabla de verdad de la función y su complemento:

A	B	C	F	F'
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Para obtener las fórmulas canónicas disyuntivas (conjuntivas) nos tenemos que quedar con las combinaciones para las que la función toma el valor '1' ('0'), y generar los mintérminos (maxtérminos) correspondientes:
F = $\sum$ m(3,4,5,6,7) = a'bc + ab'c' + ab'c + abc' + abc
F = $\prod$ M (0,1,2) = (a+b+c)(a+b+c')(a+b'+c)
F' = $\sum$ m(0,1,2) = a'b'c' + a'b'c + a'bc'
F' = $\prod$ M(3,4,5,6,7) =
(a+b'+c')(a'+b+c)(a'+b+c')(a'+b'+c)(a'+b'+c')

- F = (a + b)·(a'·c + d)
- F = a·b + b'·c + c·a'·d

2.5. Mediante los postulados y teoremas del Álgebra de Boole, obtenga unas expresiones mínimas en suma de productos de las siguientes funciones. A partir de estas sumas de productos, obtenga una expresión en producto de sumas.

- F = $\sum$m(0,1,3,4,6,8)

En primer lugar, pasamos esta expresión a suma de productos:
F = a'b'c'd' + a'b'c'd + a'b'cd + a'bc'd' + a'bcd' + ab'c'd'
Ahora observamos qué términos podemos agrupar para sacar factor común. En el caso de que haya un término que podemos agrupar con más de uno, aplicamos la ley de idempotencia y desdoblamos dicho término. No obstante, trataremos de agrupar el número mínimo de términos, es decir, no agruparemos dos términos que hayan sido agrupado antes.
F = a'b'c'(d'+d) + (a'+a)b'c'd' + a'b'(c'+c)d + a'b(c+c')d'
Aplicando el cuarto postulado del álgebra de Boole:

F = a'b'c' + b'c'd' + a'b'd + a'bd'
Para pasar a producto de sumas, debemos complementar la expresión dos veces:
F'=(a+b+c)(b+c+d)(a+b+d')(a+b'+d)
Realizamos las multiplicaciones de los paréntesis:
F'= (ad+b+c)(a+bd+b'd')
F'= ad + ab + ac + bd + bcd + b'cd'
De nuevo aplicaremos las leyes de DeMorgan:
F = (F')'=(a'+d')(a'+b')(a'+c')(b'+d)(c'+d)

- $F = \sum m(0,1,2,4)$
- $F = \sum m(0,1,6,8,10,11,12,13)$
- $F = \sum m(0,1,14,15)$

2.6. Realizar las siguientes operaciones utilizando la aritmética binaria:

- 12 + 30

Primero pasamos los números decimales a códgio binario, es decir, 12 = '01100' y 30 = '11110'. A continuación, realizamos la suma:
111000 -> acarreo
001100 -> 12
011110 -> 30
101010 -> 42

- 2+4+8

Igual que el anterior, pero primero se suman los dos primeros sumandos, y después su resultado se suma con el tercero.

- 12-3 (usando la resta binaria)

Primero pasamos los números decimales a códgio binario, es decir, 12 = '1100' y 3 = '0011'. A continuación, realizamos la resta:
1100 -> 12
0110 -> desbordamiento
0011 -> 03
1001 -> 09

- 20-10 (usando el complemento a dos)

Primero pasamos los números decimales a códgio binario, es decir, 20 = '10100' y -10 = -'01010' = '10110'. A continuación, realizamos la suma:
01000 -> acarreo
10100 -> +20
10110 -> -10
01010 -> +10

- 3·5

- $3.25 \cdot 4$

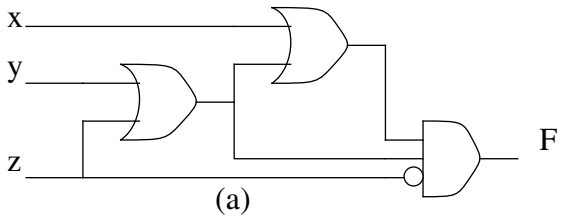
Primero pasamos los números decimales a códgio binario, es decir, 3.25 = ‘11.01’ y 4 = ‘100’. A continuación, realizamos la multiplicación:

```
             11.01 -> 3.25
             x 100 ->  x4
             00 00
            000 0
           1101
           1101.00 ->13.00
```

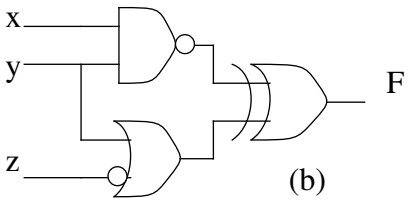
- $15.5 / 3.25$

Problemas del Tema 3

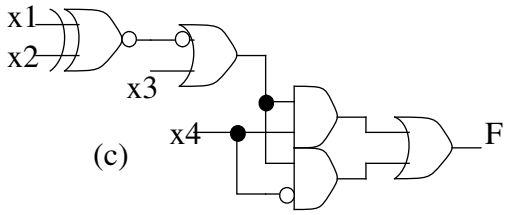
3.1. Analizar de forma estacionaria y transitoria los siguientes circuitos, así como determinar los caminos críticos:



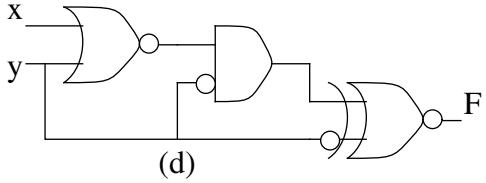
(a)



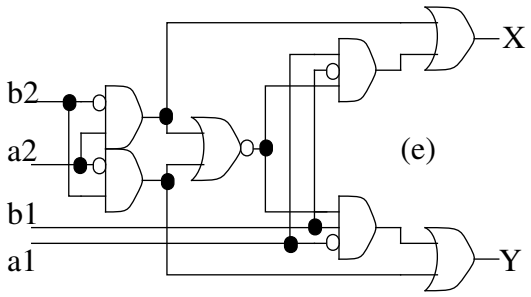
(b)



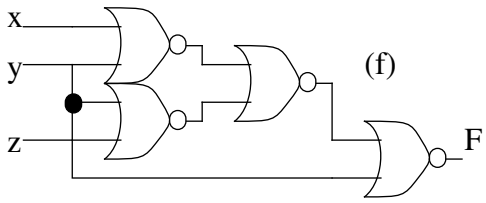
(c)



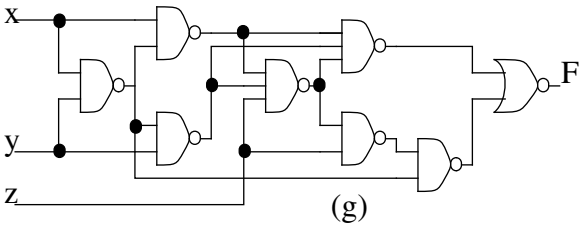
(d)



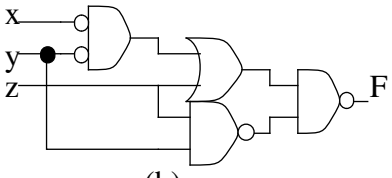
(e)



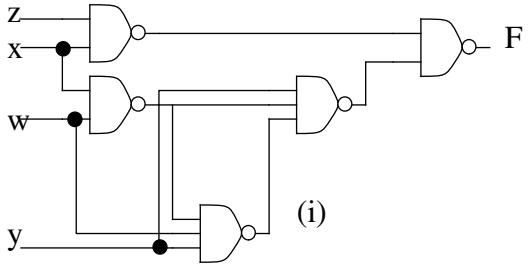
(f)



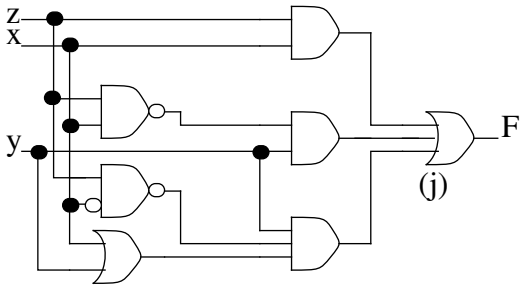
(g)



(h)



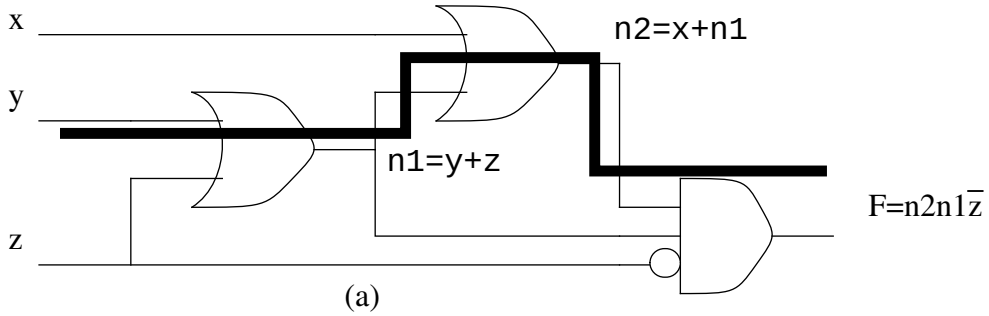
(i)



(j)

* Circuito (a)

Para obtener el análisis estacionario, etiquetamos todos los nodos con la función lógica correspondiente.



(a)

Por lo tanto, la función lógica que realiza este circuito es: **F = (x+y+z)(y+z)$\overline{z}$**.

En el análisis transitorio, tenemos que buscar alguna señal que recorra más de un camino (hasta llegar al nodo de salida), y que en alguno de ellos la señal esté complementada. En este caso particular, sólamente pueden existir azares en la señal z, ya que la señal x sólo recorre un camino, y la señal y recorre dos pero en ambos está sin complementar.

Las condiciones de azares son:

- azares estáticos:
  - F = z+$\overline{z}$
  - F = z$\overline{z}$
- azares dinámicos:
  - F = z(z+$\overline{z}$)
  - F = z+z$\overline{z}$

Por simple inspeción de la fórmula, podemos observar que sólo se pueden dar azares estáticos de la forma z$\overline{z}$. Para que se produzcan estos azares, las combinaciones del resto de señales deben ser:

- **x=1 y=0 --> F = $\overline{z}$z**
- **x=0 y=0 --> F = $\overline{z}$zz**

El camino crítico es aquel que muestra un retraso mayor. Como no se nos da el retraso de cada puerta, suponemos que todas las puertas tienen el mismo retraso, por lo que el camino crítico será aquel que atraviese más puertas. En nuestro caso, dicho camino será OR - OR - AND. Las combinaciones que siguen este camino son:

- Para que la última puerta AND espera a la puerta OR del camino, se tiene que cumplir:
  - $\overline{z}$=1 --> z=0
  - y+z=1
- Para que la segunda puerta OR espere a la primera se tiene que cumplir:

- x=0

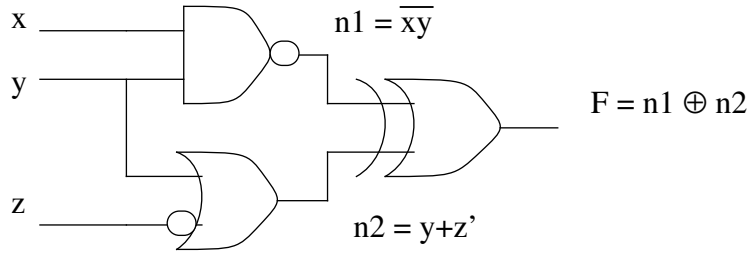
Las combinaciones que cumplen todas estas condiciones son:

- x=0, y=1, z=0

Como dicho camino es seguido por alguna combinación de entradas, el camino anterior era el crítico, formado por tres puertas.

* Circuito (b)

Para obtener el análisis estacionario, etiquetamos todos los nodos con la función lógica correspondiente.



Por lo tanto, **F = (xy)' ⊕ (y+z')**

En el caso del análisis transitorio, veamos como influyen las puertas XOR en el caso de los azares:

- $x \oplus x = xx' + x'x = xx' \dashrightarrow 0$
- $x \oplus x' = xx + x'x' = x + x' \dashrightarrow 1$

Entonces, la operación XOR de la misma señal (por dos caminos diferentes) pueden provocar azares, aunque ambos caminos sean complementados o sin complementar. Centrándonos en nuestro caso, la única señal en la que se pueda presentar algún azar es la señal y, por ser la única que sigue más de un camino.

- F = y' ⊕ y, si x='1' y z='1'

Por lo tanto, existe un único azar en la señal y, cuando x=z='1'.

Analicemos la influencia de la puerta XOR en el camino crítico:

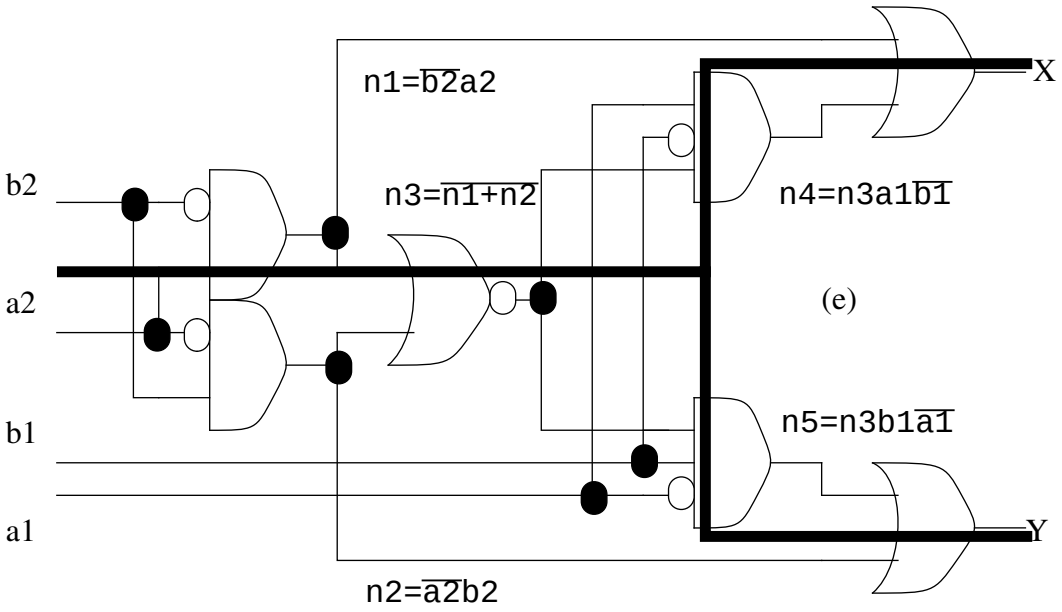
- $F = x \oplus y = xy' + x'y$

Por lo tanto, independientemente del valor lógico de alguna de sus entradas, el resultado de la puerta depende de la puerta restante, así que estas puertas no introducen ninguna restricción.

En nuestro caso particular, encontramos dos caminos críticos: NAND-XOR y OR-XOR. Como la última puerta es una XOR, su resultado siempre va a depender del nivel anterior. Por lo tanto, esos serán los caminos críticos que presenta nuestro circuito, dándose para todas las combinaciones de entrada.

* Circuito (e)

Para obtener el análisis estacionario, etiquetamos todos los nodos con la función lógica correspondiente.



Por lo tanto, el valor de las funciones de salida X e Y son:

- X = $\overline{b2}$a2 + $\overline{b1}$a1($\overline{b2'a2+b2a2'}$)
- Y = $\overline{a2}$b2 + b1$\overline{a1}$($\overline{b2'a2+b2a2'}$)

Para el caso del análisis transitorio, debemos considerar solamente las señales que siguen más de un camino (y alguno de los cuales debe estar complementada y sin complementar), para la misma salida. Las únicas señales que cumplen este requisito son las señales b2 y a2, con dos caminos diferentes. Por lo tanto, sólo podemos buscar azares estáticos:

- Para quedarnos con b2’ en la función X, la señal a2 debe valer ‘1’
- Para quedarnos con b2 en la función X, la señal a2 debe valer ‘0’, b1 ‘1’ y a1 ‘0’

Como no existe ninguna combinación que cumpla todas las restricciones, concluimos con que no existen azares en b2 para la función X. Además como la función es conmutativa con respecto a b2 y a2, podemos concluir que tampoco habrá azares en a2.

Para el caso de la función Y, vemos que el comportamiento es exactamente el mismo que en la función X, por lo que tampoco habrá azares para esta función.

El camino crítico de una función multisalida es el camino más crítico de todas las salidas por separado. Podemos observar que el camino más largo es el mismo en ambas salidas, el cual recorre: AND2-NOR-AND3-OR.

- Para que la puerta OR espera a la AND3, se tiene

que cumplir:
    - n1 = b2’a2 = 0 para la función X
    - n2 = b2a2’ = 0 para la función Y
- Para que la puerta AND3 espere a la NOR, se tiene que cumplir:
    - b1=’0’ y a1=’1’ para la función X
    - b1=’1’ y a1=’0’ para la función Y
- La puerta NOR siempre espera a una AND2, ya que todas sus entradas son salidas de estas puertas.

Las combinaciones que siguen estos caminos son:
- a1b1a2b2 = 100-, 10-1 para la función X
- a1b1a2b2 = 011-, 01-0 para la función Y

Como existen combinaciones de las señales de entradas que siguen estos caminos, podemos afirmar que el camino crítico es el formado por **AND2-NOR-AND3-OR**.

Problemas del Tema 4

4.1. Se desea diseñar el circuito de control de una planta de montaje encargado de la señal de aviso de evacuación. Para ello se dispone de tres sensores:

- A.- sensor de incendio,
- B.- sensor de humedad y
- C.- sensor de presión

Los materiales con los que se trabaja en dicha planta son tales que son inflamables y sólo toleran unos niveles mínimos de presión y humedad de forma conjunta (estos niveles se encuentran programados en los sensores correspondientes). El circuito a diseñar debe ser tal que active una señal de alarma cuando exista riesgo para los operarios de la planta.

En primer lugar debemos identificar las entradas y salidas del circuito. Como entradas tenemos los tres sensores, mientras que como salida tenemos la señal de alarma.

En segundo lugar, tenemos que generar la función lógica que queremos diseñar. Para este cometido nos fijamos en las especificaciones del diseño:

- La alarma se activará cuando exista un incendio, es decir, el sensor A se active; ya que los materiales son inflamables.
- La alarma se activará cuando exista mucha presión y humedad, es decir, cuando se activen simultáneamente los sensores B y C; por propiedad de los materiales.

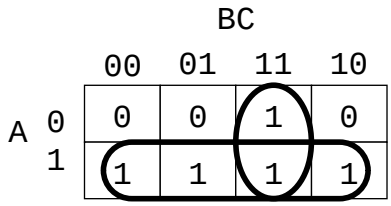
Por lo tanto, la función lógica será:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

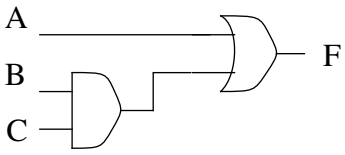
Para generar el circuito, y debido a que estamos ante una función con una sola salida, vamos a utilizar el métod del mapa.

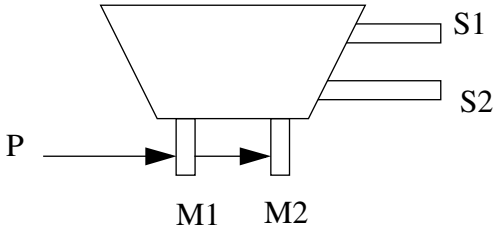
4.2. Se desea diseñar un circuito de control de una máquina trituradora. En esta máquina existen dos sensores de llenado (S1 y S2), que determinan el nivel de los elementos a triturar como se muestran en la figura:

Cuando la máquina se encuentra llena del todo, tienen que entrar en funcionamiento ambos trituradores; cuando se encuentra medio lleno, sólo tiene que funcionar uno de



F = A + BC





ellos; mientras que si no se detecta ningún elemento a triturar, ambos motores se han de parar. Dicha máquina tiene un mecanismo de emergencia a través de un conmutador de trituración, de tal forma que cuando está conectado la máquina opera según su contenido, mientras que si está desconectado, la máquina ha de pararse independientemente de su contenido.

De nuevo tenemos que identificar las entradas y salidas de nuestro sistema:

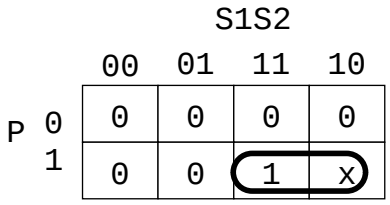
- Como entradas tendremos los dos sensores de presencia, S1 y S2, y el conmutador de trituración, P.
- Como salida tendremos las dos señales que controlan los motores de trituración, M1 y M2.

El siguiente paso consiste en obtener la tabla de verdad de la función lógica que queremos implementar:

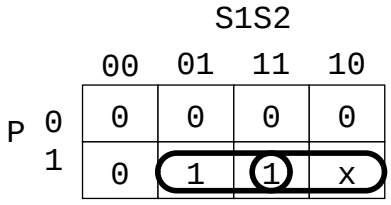
P	S1	S2	M1	M2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	0
1	1	0	x	x
1	1	1	1	1

La combinación S1 = ‘1’ y S2 = ‘0’ no se puede producir nunca en un sistema real. Por lo tanto, el valor

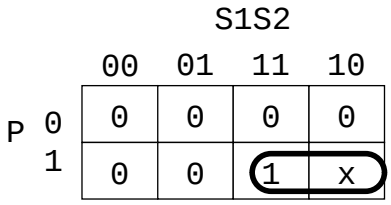
de las salidas en dicha combinación no importa y se trata como una inespecificación para tratar de reducir lo máximo posible el circuito final.
Como se trata de una función multisalida, M1 y M2, debemos utilizar el método de McCluskey. En primer lugar hallaremos los imlicantes de la función, y depués completaremos la tabla de McCluskey:



M1*M2



M1

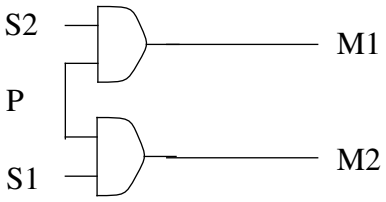


M2

M1 y M2 ---> I1 = P*S1
M1 --------> I2 = P*S2

	5	7	7
I1		x	x
I2	x	x	

M1 = P*S2
M2 = P*S1



En este caso podemos apreciar que todos los implicantes son esenciales, y deberán estar en el circuito lógico.

4.3. Se desea diseñar un circuito de interfaz binaria-decimal, de tal forma que se active una señal indicando la combinación binaria que se ha introducido a la entrada. Realizar el diseño para números codificados con dos bits, siendo este elemento lo que se conoce como decodificador 2:4.

De nuevo debemos identificar las entradas y salidas de nuestro sistema:

- Como entradas tendremos los bits cuya combinación se quiere detectar. Como en nuestro caso son dos bits, denominémosloss B0 y B1.
- Como salidas tendremos todas las posibles combinaciones de los bits de entrada. En nuestro caso serán cuatro combincaciones, denominémoslas C0, C1, C2 y C3.

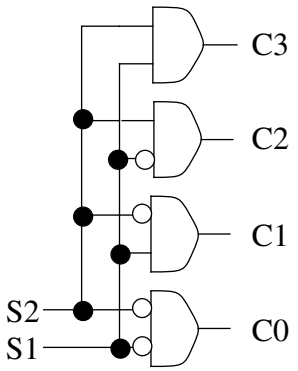
El siguiente paso consiste en obtener la función lógica que se desea imlementar. Supongamos que la salida Ci detecta la combinación i. Luego la función será:

B1	B0	C3	C2	C1	C0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Por tratarse de una función multisalida se debería aplicar McCluskey. No obstante, podemos comprobar que las salidas no tienen ningún '1' en común por lo que se pueden tratar como funciones diferentes. Además como las funciones sólo tienen un '1' y todo lo demás '0', podemos halar el término producto directamente de la tabla, ya que será el mintérmino correspondiente. Así, las salidas tendrán las siguientes fórmulas:

- C3 = B1*B0
- C2 = B1*B0'
- C1 = B1'*B0
- C0 = B1'*B0'

Luego el circuito correspondiente será:



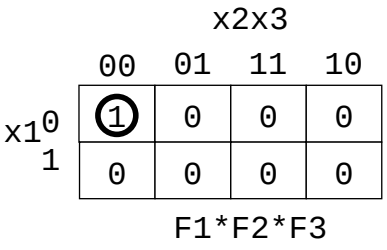
4.4. Se dispone de un código octal codificado en binario, con un bit de paridad, para la transmisión de datos entre dos estaciones espaciales. Se desea diseñar un circuito que indique la presencia de un error en un solo bit.

4.5. Encontrar los circuitos mínimos para las siguientes funciones de conmutación:

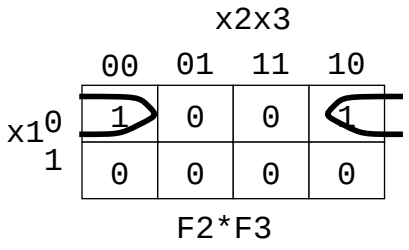
- F1 = m0 + m4 + m5
  F2 = m0 + m2 + m3 + m4 + m5
  F3 = m0 + m1 + m2

Como ya nos dan la función lógica que debemos implementar, sólo los queda hallar el circuito lógico. Esta función tendrá tres entradas, x1, x2 y x3, ya que sólo tiene hasta el quinto mintérmino. Los implicantes serán:

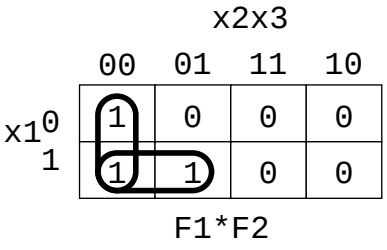
- F1, F2 y F3 --> I1 = x1'*x2'*x3'
- F1 y F2 --> I2 = x2'*x3', I3= x1*x2'
- F2 y F3 --> I4 = x1'*x3'



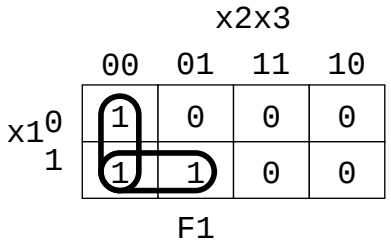
F1*F2*F3



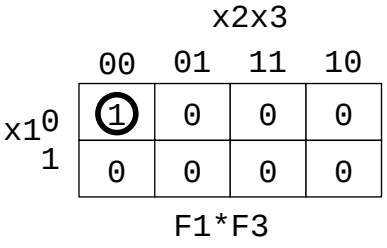
F2*F3



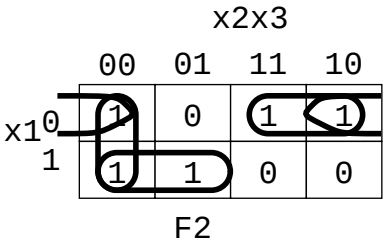
F1*F2



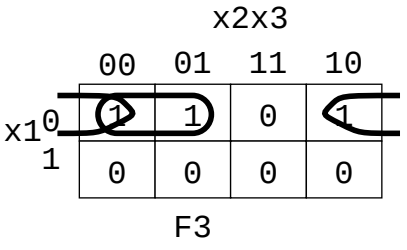
F1



F1*F3



F2



F3

- F2 --> I5 = x1'*x2
- F3 --> I6 = x1'*x2'

La tabla de McCluskey será:

	F1			F2					F3		
	0	4	5	0	2	3	4	5	0	1	2
I1	X			X					X		
I2	X	X		X			X				
I3		X	X				X	X			
I4				X	X				X		X
I5					X	X					
I6									X	X	

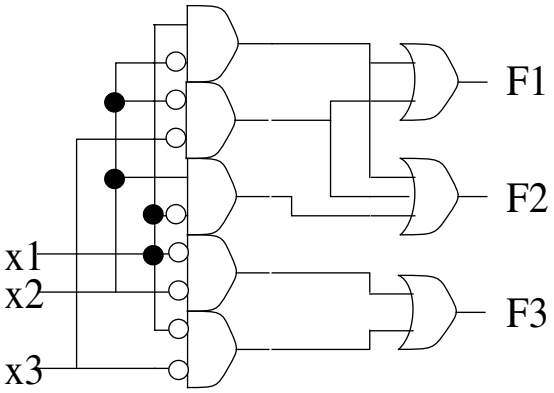
Aplicando los criterios de esencialidad, obtenemos que I3 es esencial para F1, I3 e I5 son esenciales para F2, e I6 e I4 son esenciales para F3. Eliminando estas filas y sus columnas marcadas, nos queda la tabla:

En la tabla restante, aplicamos los criterios de dominancia. Comprobamos que el implicante I4 es domi-

	F1	F2
	0	0
I1	X	X
I2	X	X
I4		X

nado tanto por I1 como por I2. También se observa que I2 e I1 son equivalentes, por lo que nos quedamos con el de menos coste. Éste será I2 por ser una puerta AND de dos entradas, frente a la de tres que es I1. Por lo tanto, las fórmulas lógicas serán:

- F1 = I3 + I2 = x1*x2′ + x2′*x3′
- F2 = I3 + I5 + I2 = x1*x2′ + x1′*x2 + x2′*x3′
- F3 = I6 + I4 = x1′*x2′ + x1′*x3′



- $F = \sum m(0,2,4,8,10,12)$
- $F = \sum m(1,4,5,7,13) + \phi(3,6)$
  $G = \sum m(3,5,7) + \phi(6)$
- $F1 = \sum m(5,7,12,13) + \phi(2)$
  $F2 = \sum m(0,1,2,5) + \phi(7)$
  $F3 = \sum m(1,2,5,12) + \phi(13)$

4.6. Realizar el ejercicio anterior, imponiendo la restricción de que todos los circuitos sean libres de azares.

- F1 = m0 + m4 + m5
  F2 = m0 + m2 + m3 + m4 + m5
  F3 = m0 + m1 + m2

La única diferencia con el caso anterior consiste en que las columnas de la tabla de McCluskey se colocan pares de mintérminos adyacentes, en lugar de solo mintérminos. Por lo tanto, los implicantes serán los mismos que en el ejercicio anterior. La nueva tabla de McCluskey será:
En este caso, todos los implicantes son esenciales excepto I1. Las fórmulas de conmutación serán:

	F1		F2				F3	
	0-4	4-5	0-2	0-4	2-3	4-5	0-1	0-2
I1								
I2	X			X				
I3		X				X		
I4			X					X
I5					X			
I6							X	

- **F1 = I2 + I3 = x2'*x3' + x1*x2'**
- **F2 = I2 + I3 + I4 + I5 = x2'*x3' + x1*x2' + x1'*x3' + x1'*x2**
- **F3 = I4 + I6 = x1'*x3' + x1'*x2'**

Problemas del Tema 5

5.1. Determinar si el conjunto formado por multipliexores forma un conjunto completo.

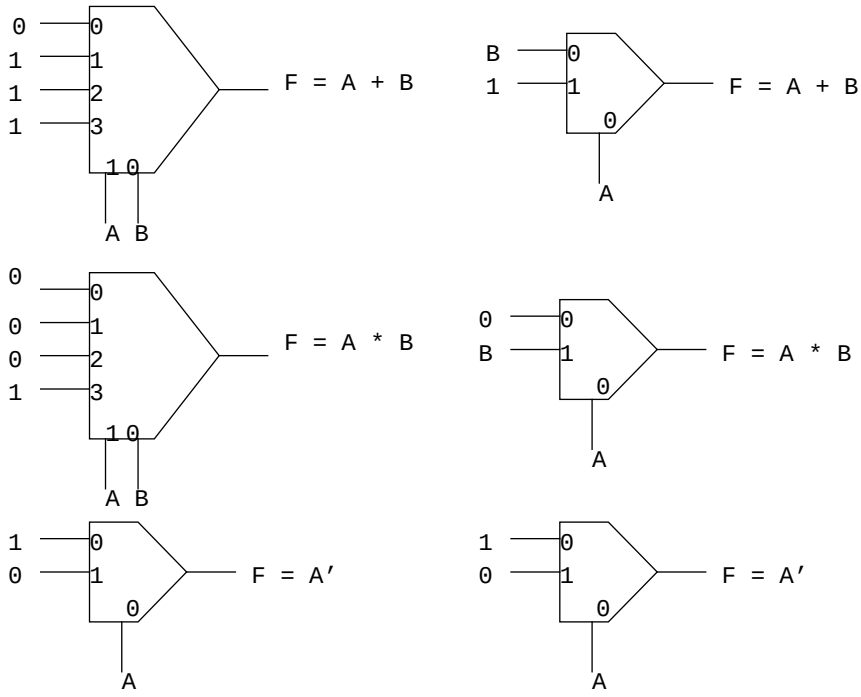
Para determinar cuando un conjunto de puertas es completo, debemos generar otro conjunto completo, como por ejemplo puertas AND, OR e inversores.

A la hora de implementar una función mediante multiplexores, la forma más intuitiva consiste en crear la tabla de la función, tomando las entradas como las señales de selección y la salida en función de las restantes entradas. Luego, cada fila de dicha tabla será cada canal de entrada del multiplexor. Por ejemplo:

A	B	F=A+B	A	B	F=A*B	A	F=A'
0	0	0	0	0	0	0	1
0	1	1	0	1	0	1	0
1	0	1	1	0	0		
1	1	1	1	1	1		

A	F=A+B	A	F=A*B	A	F=A'
0	B	0	0	0	1
1	1	1	B	1	0

Por lo tanto, a nivel de circuito, estas puertas serán:



Por lo tanto, el conjunto formado por los multiplexores forma un conjunto completo.

Como podemos apreciar, para que el diseño se realice con un solo multiplexor, éste tiene que tener como mínimo n-1 señales de selección, donde n es el número de variables de la función a implementar.

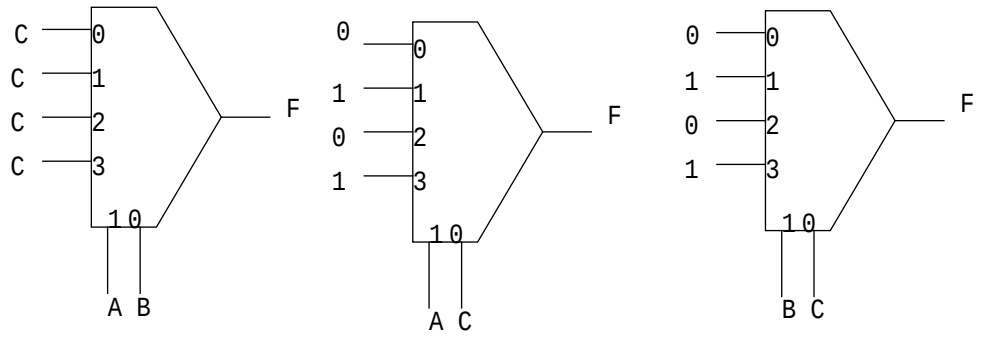
5.2. Realizar la función F= $\prod$M(0,2,4,6) mediante:

- Multiplexores 8:1
- Multiplexores 4:1

Como vimos en el ejercicio anterior, tenemos que formar la tabla de verdad de la función con dos variables de entrada (ya que nuestro multiplexor tiene dos canales de selección). Para ello nos ayudaremos viendo la tabla total de la función:

A	B	C	F	A	B	F	A	C	F	B	C	F
0	0	0	0	0	0	C	0	0	0	0	0	0
0	0	1	1				0	1	1	0	1	1
0	1	0	0	0	1	C	1	0	0	1	0	0
0	1	1	1				1	1	1	1	1	1
1	0	0	0	1	0	C						
1	0	1	1									
1	1	0	0	1	1	C						
1	1	1	1									

Las variables que elegimos para las entradas de selección serán aquellas que más nos convenga, pero no existe ningun método para indicarnos esta situación, ya que es muy ependiente de la función a implementar. A continuación mostramos todas las soluciones que hemos obtenido:

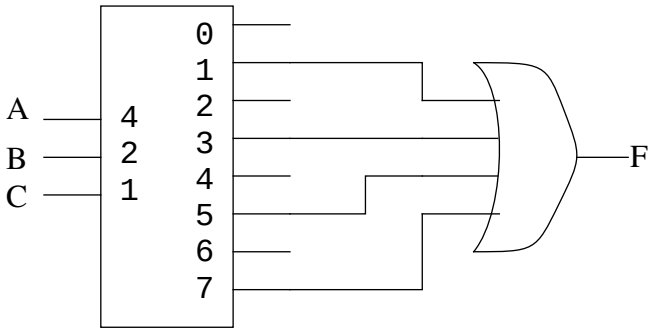


- Decodificadores 3:8 y puertas OR

En este caso, sabemos que cada canal de salida de un decodificador activo a nivel alto (bajo) identifica cada uno de los mintérminos (maxtérminos). Si lo que tenemos para unirlos son puertas OR, el único decodificador que nos sirve es activo a nivel alto; por lo que debemos pasar la función a suma de mintérminos:

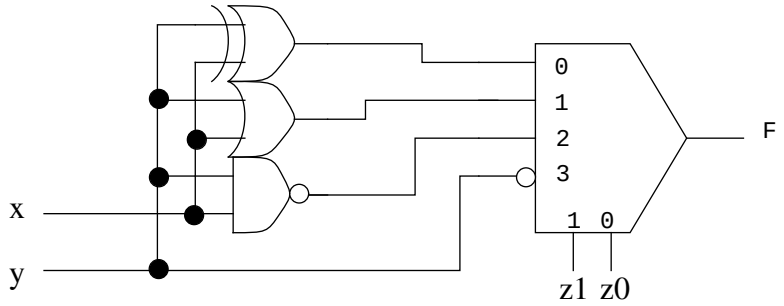
F= $\prod$ M(0,2,4,6) = $\sum$ m(1,3,5,7)

Una vez hecho esto, sólo tenemos que sumar (a través de la puerta OR) los mintérminos para los que la función vale ‘1’.



¡¡ Hay que tener mucho cuidado con el orden de las entradas del decodificador, ya que puede cambiar el índice de los mintérminos !!.

5.3. Obtener el comportamiento del circuito mostrado a continuación. Este comportamiento puede ser mostrado a través de un diagrama de flujo.



Para tratar de obtener el comportamiento de este circuito, veámoslo más detenidamente. El circuito muestra dos niveles:

- Existen una serie de puertas lógicas en el primer nivel que no están relacionadas entre sí.
- En el segundo nivel, encontramos un multiplexor que selecciona la operación de cada una de estas puertas.

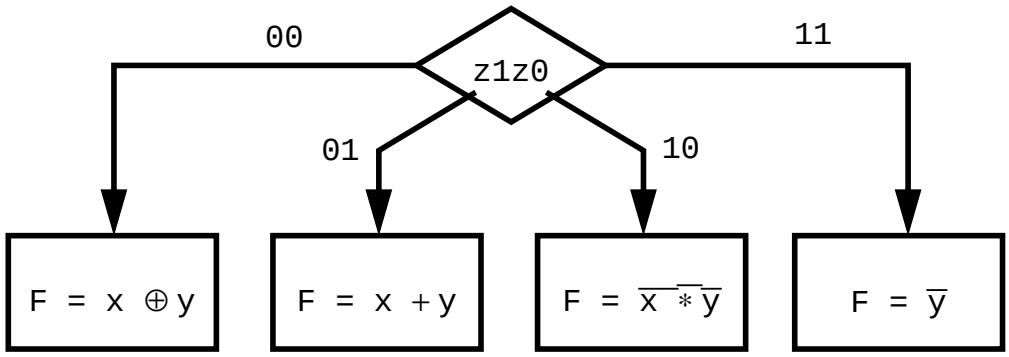
Entonces, el comportamiento de este circuito es:

- Cuando z1z0 = '00', F = x $\oplus$ y
- Cuando z1z0 = '01', F = x + y
- Cuando z1z0 = '10', F = $\overline{x*y}$
- Cuando z1z0 = '11', F = $\overline{y}$

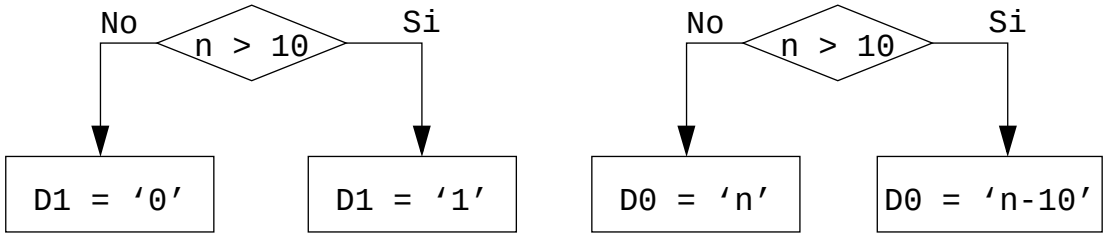
Si pasamos estas condiciones a un diagrama de flujo:

5.4. Se dispone de un dato menor o igual a 19 y siempre positivo. Se desea construir un display decimal para su visualización. Para ello se dispone de los siguientes dispositivos MSI: comparadores, sumadores/restadores y display 7 segmentos con su decodificador incorporado; utilizar el número mínimo posible.

En primer lugar tenemos que identificar las condiciones que utilizan distintos dígitos BCD. Así, cuando el número es menor que 10, el dígito más significativo (llamémoslo D1) valdrá 0, y el menos significativo (llamémoslo D0) valdrá el número de entrada. En cam-



bio, cuando sea mayor o igual a 10, el digito más significativo valdrá 1 y el menos significativo valdrá ‘n-10’. Pasemos este comportamiento a un diagrama de flujo:



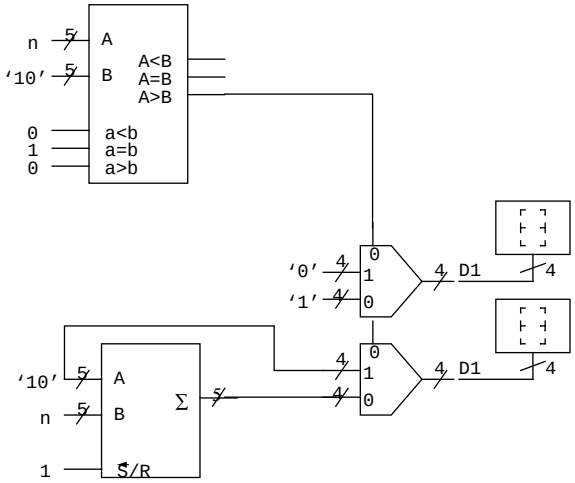
Cada componente de este diagrama tiene su equivalente en circuitos MSI. Así:

- una comparación equivale a un comparador
- una selección equivale a un circuito selector: multiplexor o demultiplexor, según el número de salidas y de entradas.
- una acción equivale a bloque que realiza dicha acción: sumador, operación lógica, ...

En nuestro caso particular necesitaremos dos multiplexores, ya que tenemos dos selecciones en las que la salida es la misma, un comparador y un sumador/restador (haciendo la función de resta). Cada multiplexor tiene una anchura de 4 bits puesto que tiene que entrar en un display 7-segmentos.

5.5. Se desea realizar un circuito que nos indique el momento y tipo de campanadas que debe dar un reloj (no el número). Los tipos de campanada son tres: cuartos, medias y horas. Para ello disponemos de los minutos en un código BCD natural (dos dígitos, M1 y M0, de cuatro bits).
El diseño ha de ser realizado con el mínimo número de las puertas siguientes: comparadores de cuatro bits y puertas lógicas.

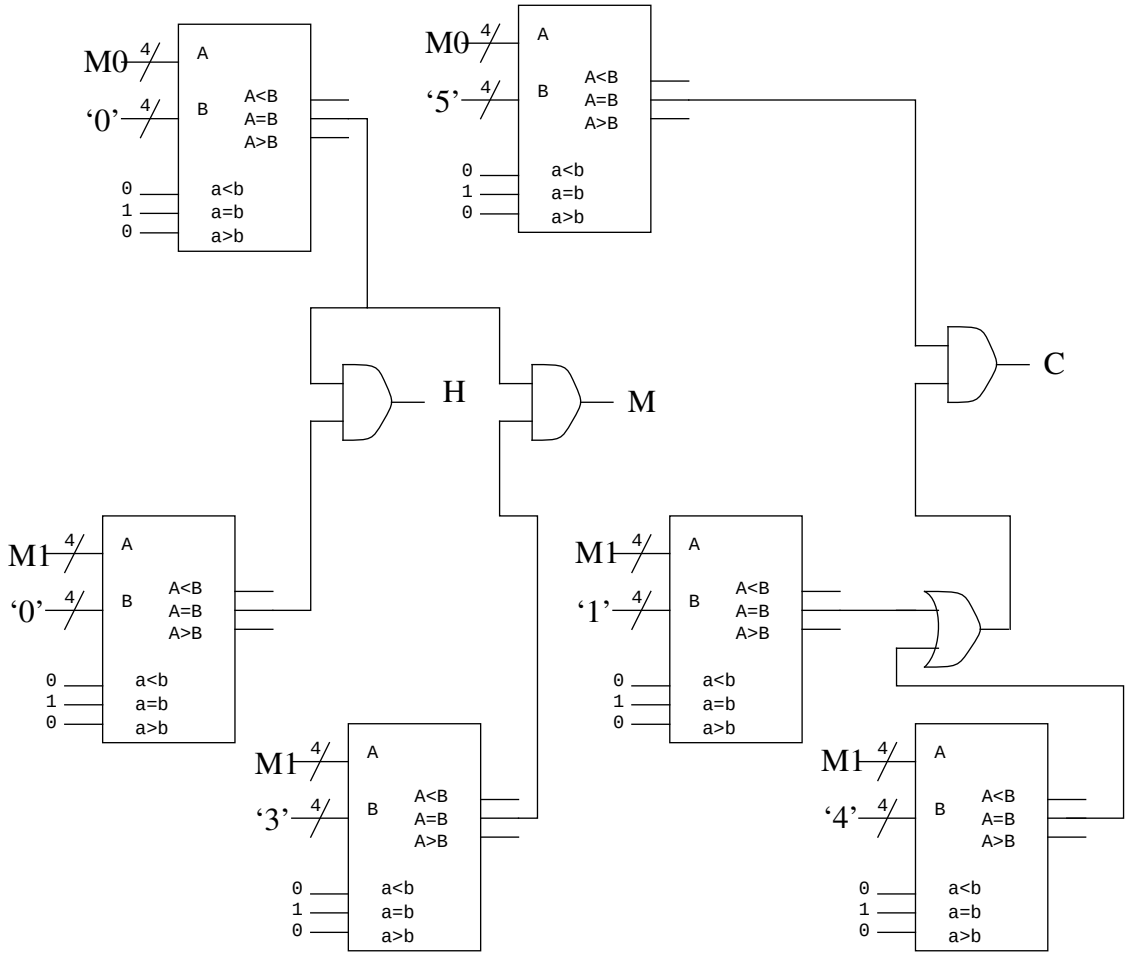
Denominemos las salidas como C(cuartos), M(medias) y H (horas). En este caso tenemos que detectar las secuencias 00, 15, 30 y 45, ya que son las que producirán las campanadas. El comportamiento del circuito



será el siguiente:

- H = 1 cuando M1M0 = '00'
- M = 1 cuando M1M0 = '30'
- C = 1 cuando M1 = '1' ó '4' y M0 = '5'

En este caso también podríamos haber hallado un diagrama de flujo, pero es más intuitivo verlo de esta manera. Luego, debemos realizar las comparaciones y unir los resultados a través de las operaciones lógicas:



Problemas del Tema 6

6.1. Realizar los diseños de los problemas del Tema 4 mediante dispositivos PROM, PAL y PLA.

De forma general, la implementación en estos tres dispositivos siguen las siguientes caracteríticas:

- Memorias ROM.- Debido al decodificador interno de dicha memoria, hay que suministrar los mintérminos para los que la función toma el valor '1'.
- Dispositivos PAL.- Debido a que las diferentes salidas no comparten ningún término producto, hay que suministrar la suma de productos mínima monosalida (todas las salidas se toman independientemente. Si hay algún término común, hay que implementarlo tantas veces como en salidas esté presente).
- Dispositivos PLA.- Debido a que todas las salidas pueden compartir todoslos términos productos, hay que suministrar la suma de productos mínima multisalida.

Ejercicio 4.1.

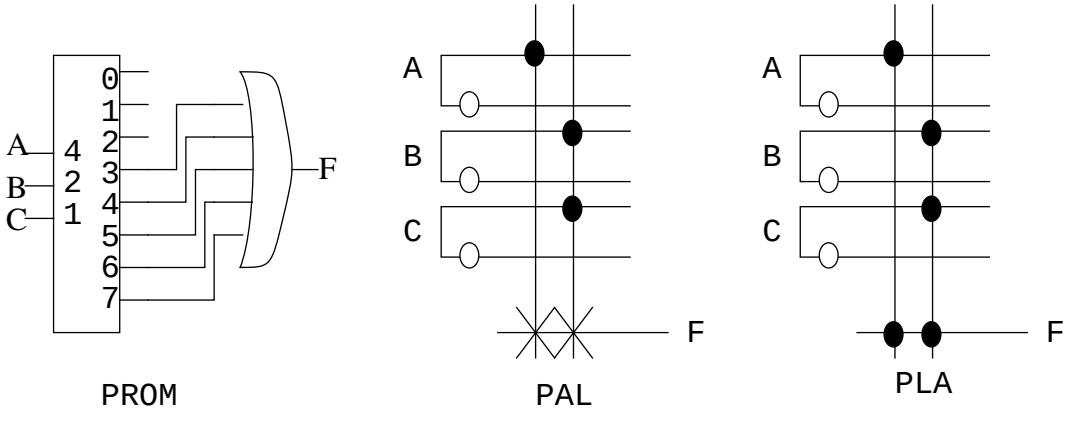
La obtención de las fórmulas se lleva a cabo de la misma forma que en el tema 4. Por lo tanto, partiremos ya de estas fórmulas.

A partir de latabla obtenida en el tema 4, podemos obtener la suma de mintérminos, y la fórmula mínima en suma de productos:

- $F = \sum m (3,4,5,6,7)$ --> PROM
  F = A + BC --> PAL
  F = A + BC --> PLA

Por lo tanto, los dispositivos tendrán las siguientes restricciones mínimas:

- ROM: 3 entradas y 1 salida
- PAL: 3 entradas, 1 salida y 2 términos AND
- PLA: 3 entradas, 1 salida y 2 términos AND

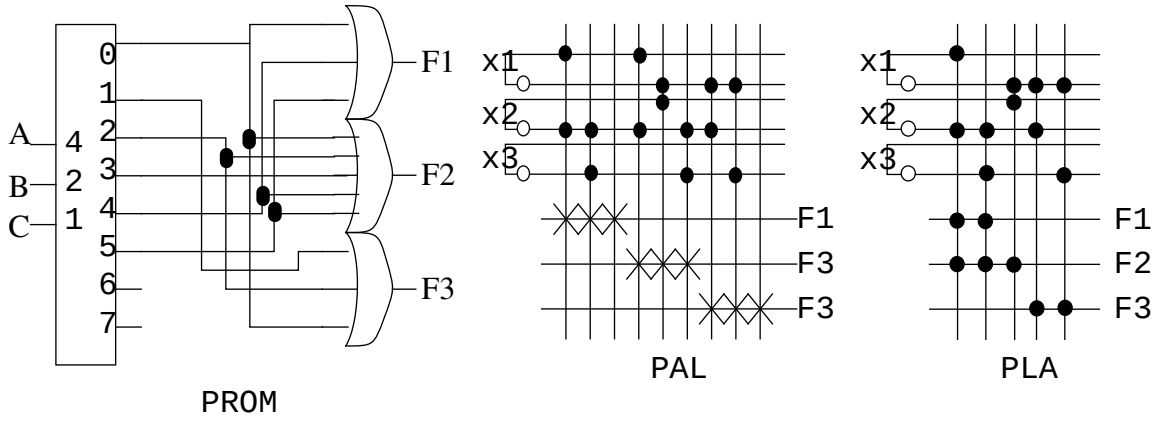


Ejercicio 4.5

- F1 = $m0 + m4 + m5$ --> PROM
  F1 = x1*x2′ + x2′*x3′ --> PAL
  F1 = x1*x2′ + x2′*x3′ --> PLA
- F2 = $m0 + m2 + m3 + m4 + m5$ --> PROM
  F2 = x1*x2′ + x1′*x2 + x2′*x3′ --> PAL
  F2 = x1*x2′ + x1′*x2 + x2′*x3′ --> PLA
- F3 = $m0 + m1 + m2$ --> PROM
  F3 = x1′*x2′ + x1′*x3′ --> PAL
  F3 = x1′*x2′ + x1′*x3′ --> PLA

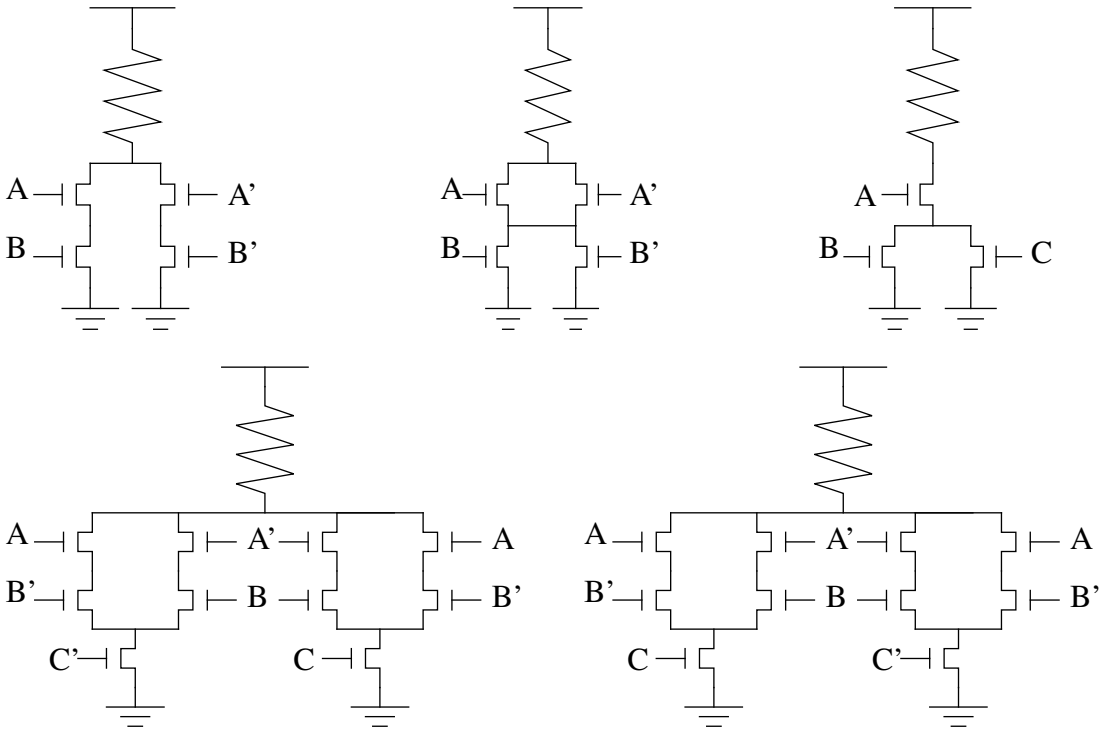
Por lo tanto, los dispositivos tendrán las siguientes restricciones mínimas:

- ROM: 3 entradas y 3salida
- PAL: 3 entradas, 3 salida y 9 términos AND
- PLA: 3 entradas, 3 salida y 5 términos AND

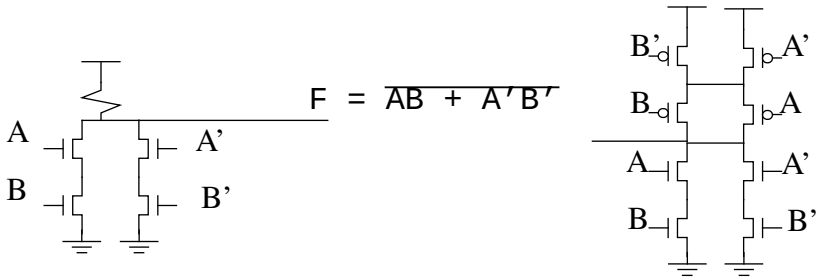


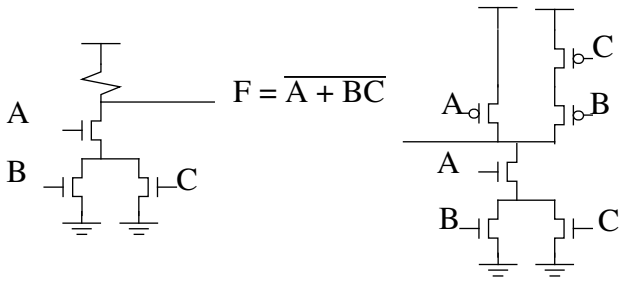
Problemas del Tema 7

7.1. Determinar la función lógica de los siguientes circuitos y realizarla mediante la lógica CMOS.

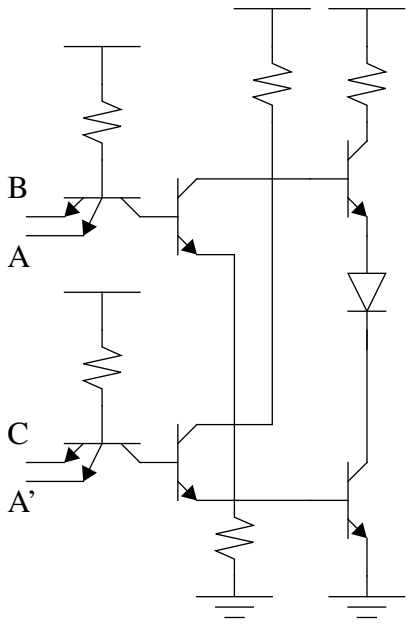


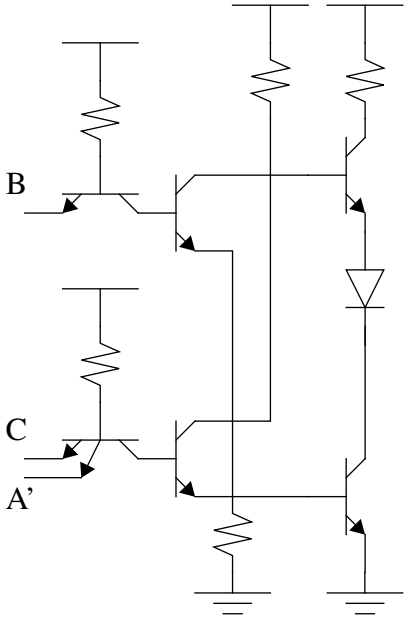
La tecnología en la que están construidas estas puertas es una tecnología NMOS. En ésta, existen transistores NMOS y un bloque de carga, formado por una resistencia en el caso qu nos ocupa. La función lógica de estas puertas es tal que la conexión en serie equivale a una operación AND, y la conexión en paralelo equivale a una operación OR, estando la salida complementada.



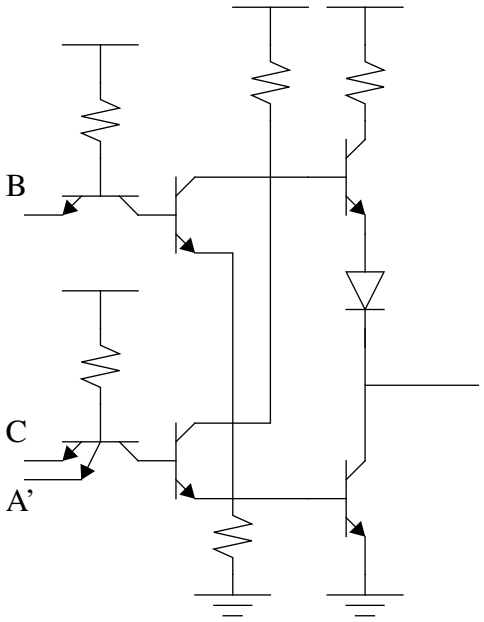


7.2. Determinar la función lógica de:





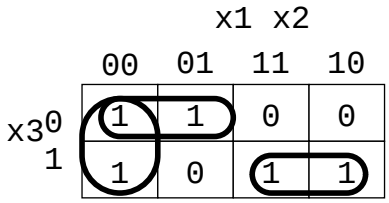
Estamos ante una tecnología TTL con salida en totem-pole. En esta tecnología (TTL), los transistores multiemisores producen la operación AND de sus entradas, y los transistores multiemisores en paralelo producen la operación OR de los productos. Y al igual que la mayoría de las familias, la salida está negada.



$F = \overline{B + A'C}$

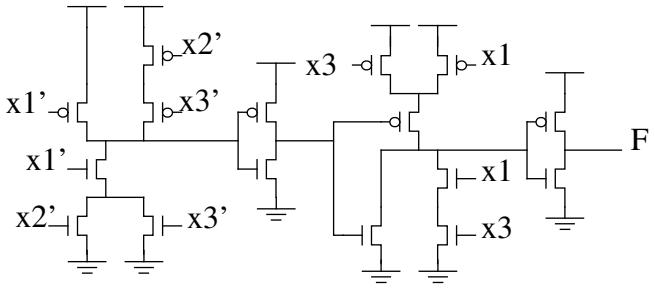
7.3. Realizar el diseño mínimo de la función f = ∑ m(0,1,2,5,7) en una familia lógica CMOS. En dicha familia, no se pueden colocar más de 2 transistores en serie de cada tipo.

En primer lugar debemos obtener la fórmula mínima que realiza dicha función lógica. Como es una función con una sola salida, aplicaremos el método de Karnaugh. Como llegamos hasta el mintérmino 7, la función tendrá tres variables de entrada



$$F(x1, x2, x3) = x1'x2' + x1'x3' + x1x3$$

Se nos ha impuesto la restricción de que solamente podemos conectar un máximo de dos transistores en serie de cada tipo, por lo que sólo podremos implementar en una puerta funciones de dos entradas. Así que haremos una puerta OR de dos productos, y su resultado se le sumará al restante.



Para reducir el número de transistores, hemos sacado factor común a x1' en los dos primeros productos. debemos notar la presencia de los inversores en la salida de cada puerta ya que las salidas son negadas. Para eliminar estos inversores, bastaría con implementar las funciones negadas de cada puerta.

7.4. Repetir el problema 7.3 para una familia TTL en la que no pueden generar transistores multi-emisores con más de 2 emisores.

7.5. Repetir el problema 7.3 para una familia ECL.

En estas familias sólo podemos implementar términos sumas (negados y sin negar). Por lo tanto, una solución sería pasar los términos productos a términos sumas complementados; quedando la función:

$F = \overline{(x1 + x2)} + \overline{(x1 + x3)} + \overline{(x1' + x3')}$

Con esta solución necesitaríamos cuatro puertas dife-

rentes, al tener cuatro sumas. Veamos si colocando la función como suma de productos, reducimos el número de transistores.

x3 \ x1 x2	00	01	11	10
0	1	1	0	0
1	1	0	1	1

$$F(x1, x2, x3) = (x1+x2'+x3')(x1'+x3)$$

$$F(x1, x2, x3) = \overline{(x1+x2'+x3') + (x1'+x3)}$$

Eligiendo esta solución, observamos que solamente necesitaremos dos puertas, así que implementaremos esta última.